MODERN VIDEO GAMES sit in a rather unique space with respect to software engineering issues: games are expected to be fun. A game that doesn’t excite, amuse, and engage a player has necessarily failed. This is very different from, say, a financial application, Web browser, or an embedded control component, all of which have more easily defined success criteria. A financial application must perform correct calculations, a browser must render HTML efficiently, and an embedded application must perform its behavior within given time and reliability budgets. For a game, the development task is to somehow ensure the outcome is indeed fun, which gives game software development and design unique properties and constraints.

The Challenge of Requirements

In a certain sense, games can also be seen as “just” large and complex software projects, unique mainly in the sense of combining several difficult and performance-sensitive software tasks—graphics, sound, interface design, artificial intelligence, networking, and so on—in a real-time context. Integrating these features and balancing their performance costs is an extremely difficult and important aspect of the actual game development. Graphics and sound, for instance, are easily recognized as defining features of most computer games, but even with modern GPUs and sound cards maximizing realism, improving rendering quality while also maximizing performance is a demanding part of game design.

Competing for CPU cycles with the graphics engine are other components...
that are equally important to providing the all-important sense of immersion that keeps players emotionally invested in games. Artificial intelligence offers players a ready and capable opponent, gives a sense of life to the many non-player characters within games, and provides partial automation, simplifying the game interface and making gameplay and interaction feel seamless. Networking, too, must be integrated; multiplayer games have become the industry standard, and larger scales of multiplayer interaction are a major attractive force for players.

The task of developing a game—of engineering fun—is at least partly one of simultaneously solving or optimizing these algorithmic and software development challenges. There is, however, another aspect to games that makes them truly unique as software tasks, and that arises from the basic fact that “fun” is a poorly specified, very abstract concept. Of course, psychological and sociological theories attempt to define pleasure, play, and related concepts, but how do these translate to actual requirements, techniques, and designs in a software project? As an engineering problem and requirement, the disconnect is quite large—anyone who has even casually watched others observing the same set of animations, that at best culminate in a brief, graphical version of “you win”?

Games somehow tap into psychological, aesthetic, and social pleasures that are extremely difficult to quantify and predict in a computer science context—the property of being addictive, providing social validation, or being fun is an imprecise, fragile consequence, lacking a clear and strong connection to the technical and software elements that actually comprise a software product.

It’s therefore unsurprising that game design and appropriate development strategies are an area of active research, both for game development companies and academics.

In This Issue

The articles we selected for this special issue showcase this variety, addressing development processes, game design itself, and testing and verification concerns.

Our first article by Andre Furtado and his colleagues focuses on the issues related to approaching games as software products. Good development practice in other software contexts typically focuses on formalization and reuse, establishing and optimizing metapatterns that allow the resulting goals are still on track. A significant amount of work in software engineering approaches to games concentrates on defining improved, formalized approaches to the development process.

The article by Alexandre Denault and Jörg Kienzle focuses on a direct technological problem, but one driven by the social aspects of computer games. The trend to massive multiplayer designs has strong appeal to players, bringing a powerful social element directly into the gameplay, but it includes both positive (friends) and negative (cheating) aspects. Accommodating this in software design, however, is nontrivial; distributed systems are complex, and games add real-time, latency-sensitive aspects, as well as the need for fault-tolerant designs that can withstand whatever behavior huge numbers of competing players produce. This is a hard enough problem for a game involving a few players, but when stretched to the truly massive scales of many modern games, it becomes extraordinarily difficult. Game companies use a wide variety of techniques to address such challenges, including multiple, separate game instances, partitioned worlds, delay-hiding animations, and so forth. Complete, integrated solutions are a hot research topic.

Once development begins, verifying that the game is achieving its “fun” goals is another major concern. Game-testing strategies typically resort to human studies, with repeated small-scale and final large-scale beta testing now an industry norm. More recent efforts have focused on continuous monitoring and evaluation, attempting to extract and quantify relevant player behaviors and experiences in order to tune the gameplay. The article by Eduardo Jiménez Chapresto and his colleagues at Black Rock Studio provides some insights into the complexity of both deciding on appropriate factors and seamlessly integrating a profiling design among all the other performance-
demanding components. These techniques have the further advantage of allowing the game to dynamically adjust to a wide variety of player behavior—games are learned environments, and maximal product reach is provided by accommodating as many skill levels as possible. The authors show how their technique improves gameplay, adapting the game AI to ensure that players experience an appropriate level of challenge.

Even once successfully tested and finally released, a game isn’t necessarily finished. Like all large, complex software, games include some number of program bugs. Because they’re designed to be progressed and (sometimes!) completed, however, crashes or flaws that prevent that are devastating to the player experience. In our final article, Chris Lewis and Jim Whitehead tackle the very difficult problem of repairing games. They propose a postrelease system that uses runtime monitoring and modification to maintain—and perhaps even improve—player experience. Automatically repairing software is a difficult task, of course; the process is helped considerably by recognizing that exact gameplay is an emergent property of the game itself, and even if automatic repair doesn’t guarantee precisely equivalent executions, it could still produce an overall fun experience.

We selected the articles included in this special issue from a very large and competitive set of submissions; the area is clearly a popular one, and we have not—and could not—hope to cover the gamut of interesting areas to explore and discuss. We hope, however, that these articles give a sense of the domain’s breadth and complexity, and also provide some insights useful to other areas of human-oriented software design.

CLARK VERBRUGGE is an associate professor in the School of Computer Science at McGill University in Montréal, Canada. He splits his focus between research into game formalization and analysis, and compiler and runtime optimization for concurrent languages. Verbrugge has a PhD in computer science from McGill University. Contact him at clump@cs.mcgill.ca.

PAUL KRUSZEWSKI is president of GRIP Entertainment, an independent Montréal-based company focusing on providing artificial intelligence solutions to the video game industry. He has more than 11 years of experience in the game industry, particularly in the field of artificial intelligence. Kruszewski has a PhD in computer science from McGill University. Contact him at paul.kruszewski@gripent.com.

ABOUT THE AUTHORS

CLARK VERBRUGGE is an associate professor in the School of Computer Science at McGill University in Montréal, Canada. He splits his focus between research into game formalization and analysis, and compiler and runtime optimization for concurrent languages. Verbrugge has a PhD in computer science from McGill University. Contact him at clump@cs.mcgill.ca.

PAUL KRUSZEWSKI is president of GRIP Entertainment, an independent Montréal-based company focusing on providing artificial intelligence solutions to the video game industry. He has more than 11 years of experience in the game industry, particularly in the field of artificial intelligence. Kruszewski has a PhD in computer science from McGill University. Contact him at paul.kruszewski@gripent.com.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.